

Optimizing Semantic Caching for Sequence Prediction and Memory-Optimized LLM Inference Scheduling

Aakash Mishra
aamishra@stanford.edu
Stanford University
Palo Alto, California, USA

Abstract

The growing prevalence of LLMs has presented a need for scalable inference platforms. Proxy-modelling for output token length presents an opportunity to schedule inference jobs more effectively using shortest-job first. We show that this method can be further optimized by using a semantic cache that uses query similarity to cache the output token length of similar prompts. We show that the accuracy of the proxy model can be boosted with semantic caching as well as a reduction in the regressor overhead. We integrate our approach with vLLM's memory optimized inference platform and show an 60% improvement in end-to-end vLLM inference time for the "facebook/opt-125m" model.

Keywords: Distributed Computing, Cloud Computing, Large Language Models, Inference Optimization, Scheduling Algorithms

ACM Reference Format:

Aakash Mishra. 2024. Optimizing Semantic Caching for Sequence Prediction and Memory-Optimized LLM Inference Scheduling. In *Proceedings of Cloud Computing (CS349D)*. ACM, New York, NY, USA, 6 pages. <https://doi.org/10.1145/nnnnnnn.nnnnnnn>

1 Introduction

Recent developments in LLMs has prompted a greater need for scalable inference platforms to support the diverse range of applications that are hosted above them [SE17]. In order to help these models scale with higher request rates and more data, we aim to improve end-to-end inference completion time and overall throughput.

A key issue with optimizing inference for LLMs is the auto-regressive nature of the models. Each output token sequence is generated sequentially and not all together [PD24]. This makes it difficult to predict the time to completion for a specific job. Most machine learning-based scheduling algorithms rely on some understanding of the size of the job, however, in this problem space we cannot assume this [Pen+18].

Many current platforms for online LLM inference support assume some level of fixed cost for the runtime of an inference job given that the hardware and model being queried is constant. In this case, the inference server is passed jobs in

a first-come first-serve basis (FCFS) [Yu+22]. There several platforms that use this framework such as ORCA, vLLM, AlphaServe, Clipper, Tensorflow-Serve, and Clockwork [Yu+22; Jia+21; Kwo+23; Cra+17]. The underlying issue with this approach is a situation where the first job in the queue takes a much longer time than the rest. This would cause job completion times to lag for smaller jobs that follow. Having some sort of proxy for understanding how long it could take to complete a job would help us to understand the nature of this issue.

To address this problem, several papers have opted for "proxy-modeling". This approach entails modeling the output token length for a given prompt in order to determine some sort of ranking for job completion time [Rom+21]. [Qiu+24] builds a proxy-model using a fine-tuned BERT and shows that we can achieve a relatively large increase in throughput from doing so. However, the model suffers from accuracy issues and general overhead of running a large BERT model everytime a query is passed ot the inference engine. In order to avoid this computation, we build a semantic cache that can store the output token lengths for semantically similar queries and utilize this for scheduling. We also integrate our approach with vLLM in order to provide a memory-optimized inference platform that can help with end-to-end performance.

Our main contributions are:

(1) A novel semantic cache design for output token length prediction. (2) An open-source integration of the SSFJ scheduler with vLLM. (3) A memory optimized end-to-end system that outperforms the accuracy and throughput of previous proxy-modeling approaches. Our code can be found here: <https://github.com/aakamishra/SemantiCache>

2 Related Work

2.1 LLM Serving Platforms

Large language models are generally assumed to be generative transformer-based models that have been deployed to a variety of applications in increasing prevalence [Zha+23]. Many of these models host millions of requests on a daily basis and that requires significant model resources for inference compute [Cha+24]. The larger issues is the auto-regressive nature of the generative model. Since there is a dependency on past inputs and prompt context, the model's final output is run in an iterative process. This process is usually not

deterministic and it depends on the type of model and the type of answer required [Liu+24].

Many papers have detailed issues with this auto-regressive nature and its affect upon the job completion times for inference serving platforms [Zha+23]. Particularly, the non-deterministic nature of the models makes it difficult to optimize a scheduler that can improve the end to end average completion time for a job. This challenge continues to affect many modern ML inference platforms [Qiu+24].

For example, ML serving systems that function as a wrapper to optimize inference jobs by managing resources, hardware, scheduling, networking and other abstractions cannot provide a deterministic order for an optimal inference schedule. This means that many platforms default to the user of first-come first-serve (FCFS) [Jin+23]. The known problem with FCFS is head-of-line blocking issues when the first queued job has a long completion time, which in turn delays the completion time of shorter jobs as well. For example in Orca, the authors add in iterative scheduling with continuous batching. However, despite these optimizations the underlying jobs are not re-ordered in terms of the arrival time [Wu+23]. Other frameworks like FastServe enable feedback for re-ordering an inference time queue. However, this feedback mechanism has difficult adapting to the non-determinism of LLMs.

Other papers like S³ use output token modelling in order to adjust the key-value (KV) cache size which can also improve the job completion time metric [Jin+23]. However, they do not apply this problem directly to scheduling. This opens them to competition from methods like vLLM which also use FCFS for memory-optimized inference [Kwo+23]. In order to add this memory-optimized impact in our own results, we integrate our reordering process with the vLLM serving framework as well to record the improvements in performance end-to-end.

The latest state-of-art for proxy modeling involves using a Bert based regressor model to predict the output-token length as seen in [Qiu+24]. However, there are severe accuracy issues with this modeling approach which the largest jobs having an almost 0% accuracy and non-negligible overhead for the regressor. We aim to improve upon this method by using semantic caching.

Semantic caching, as introduced by GPTCache [Ban23] involves using an embedding model to create embedding of the inference queries and then using those embedding as keys while saving the model output as the values to the semantic cache. Our approach is similar, but rather than storing just the generated output, we also make sure to store the output length, which we can then use as a proxy for future requests.

3 Design

At a high level overview the system we present builds off of [Qiu+24]. We first take the user prompt and then pass it through our encoding model. Then we take our encoding and check the cache for whether there is a hit or not with our encoding. We use cosine similarity to check the cache hit and use a threshold of 0.95 or more. If there is no semantic cache hit, then we pass the prompt to the token length predictor which then generates a predicted value for the token length. The scheduler takes a batch of prompts and then reorders them based on the shortest-job first schedule (SJFS) algorithm. The batch is then processed by the vLLM engine and returned back to the users. For each cache miss, we log the final output token length from the inference engine and add it to our cache.

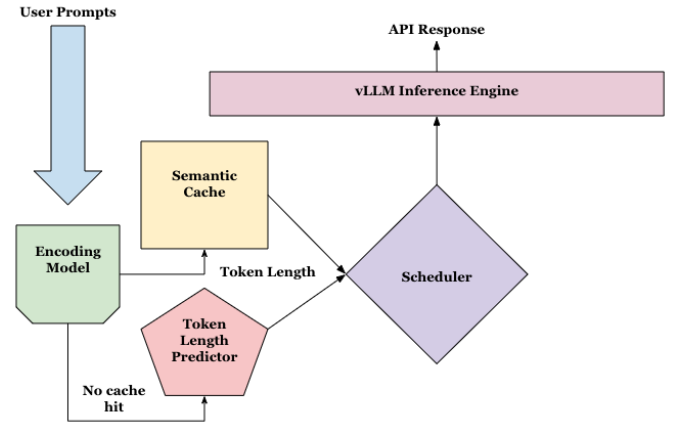


Figure 1. Semantic-Cache and Inference Overview

3.1 Encoding Model

In order to implement our semantic caching model, we first train an encoder to properly embed the user given prompts from the model. We chose to avoid using the default BERT-uncased model and we opt for a more rigourously trained encoding model that has a smaller architecture than the original BERT. If we opted for the original BERT model then we would have encountered sparsity in the pooling layer of size 768 which is often a problem for default BERT based models. The other issue we wanted to avoid is model overhead. Since the encoder will be running for every input, it is ideal to minimize the model size and overhead as much as possible. The model is trained based on a variation of the sentence-BERT paper and involves adding multiple negative ranking loss with cosine similarity to the original BERT objective function.

$$\mathcal{L} = \frac{1}{N} \sum_{i=1}^N \left[-\log \frac{\exp(\text{sim}(e_i, e_i^+))}{\sum_{j=1}^N \exp(\text{sim}(e_i, e_j^-))} \right]$$

We then fine-tune this model on the Wiki text [KV09], Stack Exchange [JF23], Squad 2.0 [RJL18], PAQ [Lew+21], and Quora Questions [Wan+13] in order to further improve the embedding quality. The model’s 368 size embeddings are then used to represent each prompt that passes through our system.

3.2 Semantic Cache Implementation

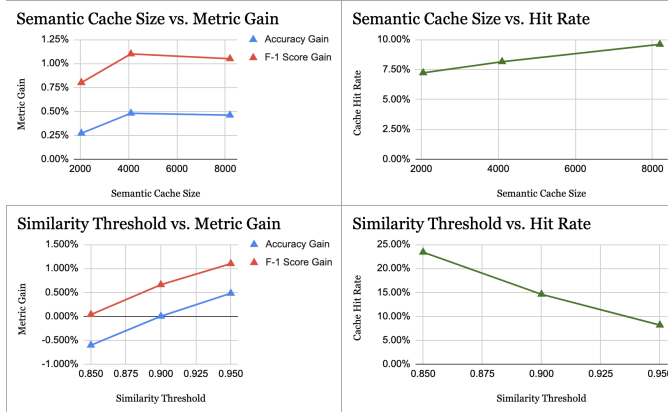


Figure 2. Semantic-Cache Hyperparameters

Our semantic cache is implemented as a mapping between the embedding value for a prompt and the model name and output token length of the query. The cache operates a least frequently used (LFU) policy and removes the last used cached embedding when the cache size is at its limit. The main difficulties with the use of a semantic cache is fine-tuning the parameters of the semantic cache. For example, if the threshold is too low, then that affects the accuracy of the model and then also the final throughput and performance. Additionally, if the cache size is too large then the overhead is affected since the number of embeddings to cross-check cosine similarity with scales linearly as well. Additionally, depending on the system there may be additional memory constraints on the cache size. In our final experiments, we tested with a cache size of 4096 and a threshold of 0.95. We chose these values after running some ablation using model accuracy as a metric to help tune our hyper-parameters using the default BERT encoder from [Qiu+24].

We can see from figure 2 that there is a point after which there is no longer an accuracy gain or an F1-score gain from increasing the cache size on our validation dataset. Additionally, if we lower the threshold by too much then we sacrifice accuracy as well in the process. Additionally, cache size does not have as much of an effect on hit rate as much as the similarity threshold does. We find this metric to be the most important to tune. From our results, we also see that there is very little operating space for us to improve the existing implementation because if we increase the threshold too much then the hit rate dramatically decreases. Essentially

this implies that the quality of our encoding model restricts our ability to further optimize the cache since the added overhead is questionable if the hit rate drops below 10%.

3.3 vLLM Integration and Scheduling

We choose to first pass each batch of prompts to our short-job first scheduler before we pass the prompts to the vLLM engine. The vLLM inferences engine uses a FCFS policy, therefore it is logically similar to order the jobs before we pass them to vLLM in order to avoid changing the module code. Once we order each batch of jobs and we run them through the inference engine, we can go back to each cache miss and then update the cache with the new hit data based on the real output length from the model. This allows our cache to adapt in real-time and helps ground the predictions for the model.

4 Evaluation

We evaluate our model performance on the LMSYS-Chat-1M dataset which contains over a million conversations with more than 20 different LLMs. We run our evaluation on a Tesla 60 NVIDIA GPU using an AWS g4dn 2xlarge instance. We first evaluate the accuracy of the model and compare it across different all 25 models. We choose different category buckets than [Qiu+24] since the previous buckets chosen were not granular and did not reveal the deficiencies with the previous proxy model. Instead we opt for a 10 bucket approach, where each bucket category represents a decile in the output token distribution. We then simulate the approximate throughput and job completion time based on simulated arrival rates of jobs given the accuracy of our model. Finally, we test our system end-to-end with a timed vLLM completion run comparison.

4.1 Comparing Model Accuracy with Semantic Caching

Using a cache setting of 4096 for cache size and a threshold similarity of 0.95, we see a 2% gain in accuracy for the model in the 10 bucket evaluation framework where the previous model only led to a 17.1% accuracy, and ours was at 19.3%. Additionally, we see that the F-1 score increased by 3% from 16.5% to 19.6% when we added the semantic cache as seen in table 1. We can further analyze the differences by category between the models using heat maps.

	Model Accuracy	Model F1-Score
Default BERT	17.1%	16.5%
S-BERT with Cache	19.3%	19.6%

Table 1. Gains from adding a Semantic Cache for 10 Bucket Evaluation Across 20 Models

Each of the heatmaps presented in figure 3 are generated by visualizing the prediction matrix. Ideally, a perfect model

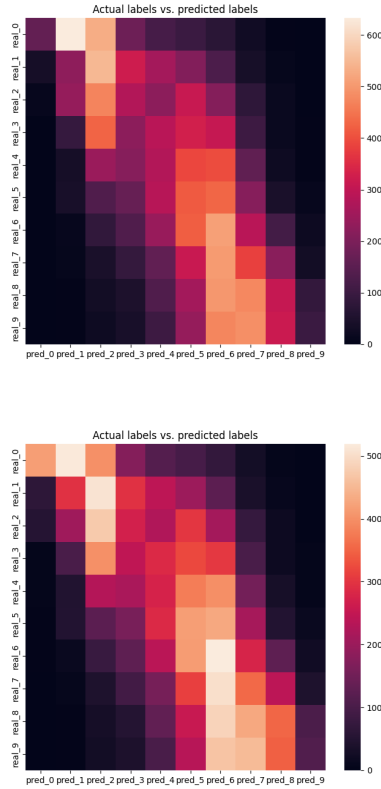


Figure 3. Heat Map Comparison for 10-Bucket Accuracy. Top image is with no cache, bottom is with caching enabled.

would only populate the diagonal of this matrix where the predicted value and real value agree. We see from these heatmaps that there is an improvement in predictions for the model when output token length is larger. More specifically, we see in the no-cache settings that there is less of a diagonal trend than in the cache usage setting. Specifically there is an uptick in predictions between the 60th to 80th percentile region. Larger prompts and answers are also more likely to have cache hits because they are more likely to produce a richer embedding that can meet the similarity threshold. Therefore, this result is in line with what we expected from our model accuracy evaluation. One clear area for improvement would be category 9 where extremely large tokens in the 90th percentile are categorized. These token lengths are likely the hardest to predict since the nature of the model becomes more non-deterministic with the size of the iteration. Overall, semantic caching seems to improve the model performance in a crucial area by improving accuracy in longer token categories.

4.2 Scheduling Results

The next step of our evaluation was to measure the effect of the accuracy increase on the actual scheduling throughput

and job completion time given different numbers of jobs and job arrival rates under the assumption of a poisson distribution. We present these results for GPT-4, Vicuna-33B and Llama-13B. These models had the discernible changes in their pipeline throughput from relatively small changes in accuracy and F-1 score.

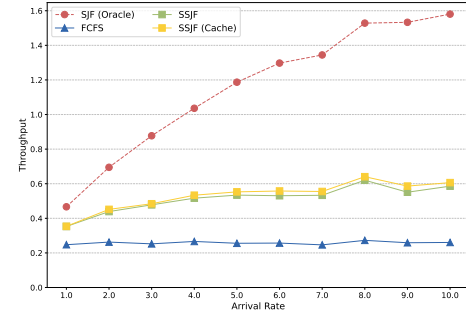


Figure 4. Vicuna-33B Throughput vs. Arrival Rate

We can see from figure 4 that there is a clear gain in adding the semantic cache on throughput but there is still a large gap between the performance of the Oracle and our system. We see that the Oracle scales differently with arrival rate than the predictor based scheduler. It appears that our caching has a linear effect on the performance of the scheduling throughput.

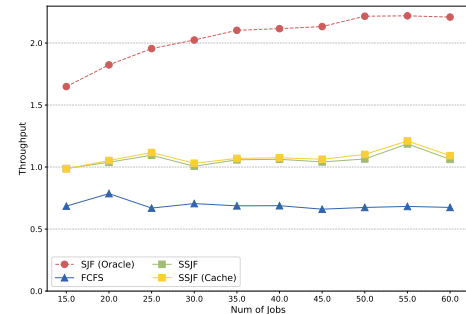


Figure 5. Llama-13B Throughput vs. Number of Jobs

In figure 5 we can also see a gain in the throughput when compared to scaling the number of jobs for the Llama-13B model. However, despite the increase in accuracy, especially for larger jobs the actual gain in throughput is relatively smaller than expected. We see that the semantic cache version performs slightly better than the default version as presented in the original paper. There is also a large gap between the Oracle and our work. However, here it does not appear that the Oracle scales that differently than the other versions as compared to the previous figure when comparing throughput to arrival rate. We estimated using random noise on the prophet that the model would have to have an

accuracy of close to 80% in order to perform as well. Since we know that the current semantic cache model has a 19.6% accuracy, these results are also relatively in line with what we expected.

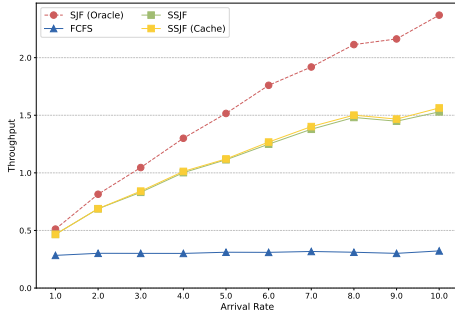


Figure 6. GPT-4 Throughput vs. Arrival Rate

We see in figure 6 that the semantic cache enabled proxy model once more performs very similarly to the default implementation. However it is slightly better than the regular BERT regressor. Additionally we see that for higher and higher arrival rates, the two curves begin to slightly diverge. Additionally, we see that here the curve for the proxy-model prediction method scales more similarly to the Oracle as compared to FCFS. Overall, the actual gains on throughput for the semantic cache are not significantly large.

4.3 Evaluating in vLLM End-to-End with Semantic Caching

In order to test the full end-to-end process of our system, we add our scheduling modifications to the vLLM inference engine by passing in an ordered set of prompts based on Oracle ordering, default BERT regressor ordering, and Semantic Cache + BERT regressor ordering. We time the beginning till end of the vLLM processing of prompts and compare the different timings for each of the schedulers. Due to constraints in GPU size, we were only able to test the end-to-end vLLM pipeline for one of the models in the LLM conversations dataset. We generated outputs for "facebook/opt-125m" which can fit on a NVIDIA Tesla 60 GPU. We took 100 random prompts for each test job and sorted them by each scheduler. We then measured the time it would take for each system to process the prompts, sort them and then output the generated content.

We are able to measure the end-to-end results in table 2. Note that we attempted to run the same generation with Huggingface generation pipelines but each of the jobs took more than 10 seconds and was not comparable in magnitude to these results. We see that the Oracle scheduler has the fastest end-to-end speed and generation speed as well. Our approach closely follows with a comparable performance that was 0.017 seconds and 37 tokens / second slower. Our semantic cache method also outperforms the default BERT

	End-to-End Time (s)	Gen-Speed (tok/s)
Oracle	0.47	1580.34
Default BERT	0.49	1519.23
S-BERT w/ Cache	0.487	1547.23
No Sorting	0.791	923.1

Table 2. End-to-End Measurement for vLLM Integration on "facebook/opt-125m"

regressor method as well. This result is as expected because cache hits should reduce overhead and the model is more accurate. Additionally we see that without any proxy model or sorting there is a clear drop in performance from 0.487 seconds to 0.791 seconds. Additionally there is a more than 600 tokens / second drop as well. This shows that by adding proxy-modelling on top of vLLM, we are able to achieve a **60% boost** in performance for the "facebook/opt-125m" model.

5 Discussion

We are able to show that by incorporating the Semantic cache into our model inference platform for LLMs, we were able to improve performance, gain higher throughput for larger models, and improve integration with vLLM while also reducing overhead costs. The difference in overhead for the default BERT method was on average 0.01700 seconds per prompt, where as it was 0.01672 seconds for the regressor with semantic caching added. This is a reduction of -0.00028 seconds. Overall the semantic caching approach shows promising results and it would be worthwhile to explore this further with a more robust encoding model. The current issue is that the cache hit rate is low and the similarity threshold has to be set high in order to avoid reduction in overall accuracy. This limits the ability of the semantic caching model to make a sizeable difference in the actual performance. By creating a more efficient encoding model, we can set a lower similarity threshold in the future and achieve a higher cache hit rate.

Future work can be continued for enabling semantic caching for multi-round conversations. Multi-round prompt embeddings can be transformed into a sequential state encoding problem which can also help with multi-round output token proxy prediction. More work can also be done with more hardware compute resources. Due to GPU limitations we only tested one model end-to-end with vLLM. However, in the future, with larger GPUs we can also generate results for Llama-13B and Vicuna-33B which showed signs of improvement in throughput using semantic caching. Additionally, since the model only needs to output scores that can help SJFS rank the prompts, it may be better to structure this as a ranking problem rather than a regression problem because the ranking approach can help generalize the model by enabling scale in-variance.

References

- [Ban23] Fu Bang. “GPTCache: An open-source semantic cache for LLM applications enabling faster answers and cost savings”. In: *Proceedings of the 3rd Workshop for Natural Language Processing Open Source Software (NLP-OSS 2023)*. 2023, pp. 212–218.
- [Cha+24] Yupeng Chang et al. “A survey on evaluation of large language models”. In: *ACM Transactions on Intelligent Systems and Technology* 15.3 (2024), pp. 1–45.
- [Cra+17] Daniel Crankshaw et al. “Clipper: A {Low-Latency} online prediction serving system”. In: *14th USENIX Symposium on Networked Systems Design and Implementation (NSDI 17)*. 2017, pp. 613–627.
- [JF23] Julie Jiang and Emilio Ferrara. “Social-LLM: Modeling User Behavior at Scale using Language Models and Social Network Data”. In: *arXiv preprint arXiv:2401.00893* (2023).
- [Jia+21] Jingyan Jiang et al. “Joint model and data adaptation for cloud inference serving”. In: *2021 IEEE Real-Time Systems Symposium (RTSS)*. IEEE. 2021, pp. 279–289.
- [Jin+23] Yunho Jin et al. “: Increasing GPU Utilization during Generative Inference for Higher Throughput”. In: *Advances in Neural Information Processing Systems* 36 (2023), pp. 18015–18027.
- [KV09] Markus Krötzsch and Denny Vrandečić. *Semantic wikipedia*. Springer, 2009.
- [Kwo+23] Woosuk Kwon et al. “Efficient memory management for large language model serving with pagedattention”. In: *Proceedings of the 29th Symposium on Operating Systems Principles*. 2023, pp. 611–626.
- [Lew+21] Patrick Lewis et al. “PAQ: 65 million probably-asked questions and what you can do with them”. In: *Transactions of the Association for Computational Linguistics* 9 (2021), pp. 1098–1115.
- [Liu+24] Mingdao Liu et al. “APAR: LLMs Can Do Auto-Parallel Auto-Regressive Decoding”. In: *arXiv preprint arXiv:2401.06761* (2024).
- [PD24] Konstantinos Papaioannou and Thaleia Dimitra Doudali. “The Importance of Workload Choice in Evaluating LLM Inference Systems”. In: *Proceedings of the 4th Workshop on Machine Learning and Systems*. 2024, pp. 39–46.
- [Pen+18] Yanghua Peng et al. “Optimus: an efficient dynamic resource scheduler for deep learning clusters”. In: *Proceedings of the Thirteenth EuroSys Conference*. 2018, pp. 1–14.
- [Qiu+24] Haoran Qiu et al. “Efficient Interactive LLM Serving with Proxy Model-based Sequence Length Prediction”. In: *arXiv preprint arXiv:2404.08509* (2024).
- [RJJ18] Pranav Rajpurkar, Robin Jia, and Percy Liang. “Know what you don’t know: Unanswerable questions for SQuAD”. In: *arXiv preprint arXiv:1806.03822* (2018).
- [Rom+21] Francisco Romero et al. “{INFaaS}: Automated model-less inference serving”. In: *2021 USENIX Annual Technical Conference (USENIX ATC 21)*. 2021, pp. 397–411.
- [SE17] Reid G Smith and Joshua Eckroth. “Building AI applications: Yesterday, today, and tomorrow”. In: *Ai Magazine* 38.1 (2017), pp. 6–22.
- [Wan+13] Gang Wang et al. “Wisdom in the social crowd: an analysis of quora”. In: *Proceedings of the 22nd international conference on World Wide Web*. 2013, pp. 1341–1352.
- [Wu+23] Bingyang Wu et al. “Fast distributed inference serving for large language models”. In: *arXiv preprint arXiv:2305.05920* (2023).
- [Yu+22] Gyeong-In Yu et al. “Orca: A distributed serving system for {Transformer-Based} generative models”. In: *16th USENIX Symposium on Operating Systems Design and Implementation (OSDI 22)*. 2022, pp. 521–538.
- [Zha+23] Wayne Xin Zhao et al. “A survey of large language models”. In: *arXiv preprint arXiv:2303.18223* (2023).